TERADYNE

321 Harrison Avenue
Boston, MA 02118

August 23, 2000

ASSISTANT COMMISSIONER FOR PATENTS

Washington, D.C. 20231
Attn: PATENT APPLICATIONS

CERTIFICATE OF MAILING BY EXPRESS MAIL
MAILING LABEL NUMBER: EL551056461US
DATE OF DEPOSIT: August 24, 2000
I certify that the attached paper or fee is being deposited with the United States Postal Service as Express Mail addressed to the Assistant Commissioner of Patents, Washington, D.C. 20231.
Elizabeth S. Clemente
signature of person mailing the paper
Elizabeth S. Clemente

Sir:

TRANSMITTAL FOR PATENT APPLICATION

Transmitted herewith for filing is a new patent application entitled:

"Method of Providing Software Testing Services"

by inventor(s):

BAKER, Caren H.
FRIEDMAN, George
GLIK, Michael V.
VAHEY, Walter G.

Enclosed are:

12	pages of specification
3	pages of claims
1	pages of abstract
1	sheet(s) of drawing(s)

FEE CALCULATION

Basic Filing Fee   690

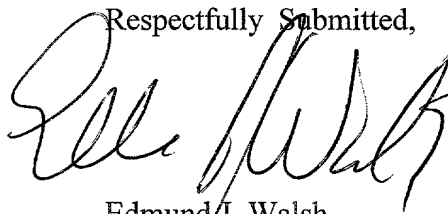|	Claims Filed	Excess Claims	Charge Per Claim	Totals
Total:	19	0	18	0
Independent:	1	0	78	0

Total Fee: $ 690

Please charge the Total Fee to Teradyne, Inc.'s Deposit Account No. **20-0515**.
The Commissioner is hereby authorized to charge fees under 37 CFR 1.16 and 1.17
which may be required, or credit any overpayment to Deposit Account No. 20-0515. One
additional copy of this sheet is enclosed.

Please date stamp and return the enclosed self-addressed, stamped post card to
evidence receipt of this application.

Respectfully Submitted,

Edmund J. Walsh
Attorney for Applicant
Reg. No. 39,250

Atty. Docket No.   : 1406-US
Telephone   : 617-422-2853
Fax Number   : 617-422-2290

TITLE OF THE INVENTION

Method of Providing Software Testing Services

5

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority from provisional US application 60/151,418 filed August 30, 1999, and U.S. Application 09/482,178 filed on January 12, 2000, for Method and System for Software Object Testing, which is hereby incorporated by reference.

10

BACKGROUND OF THE INVENTION

This invention relates generally to computer software applications and more specifically to testing computer software applications.

A traditional client/server application is run in an enterprise that includes several client computers such as individual workstations. These workstations are connected, typically over a network to a central computer that is programmed with an application. The application program contains the business logic – or the instructions to perform the steps needed to make the application useful to the business that is using it. For example, the business logic for an application used by an insurance company would define such things as what types of insurance policies could be issued or the cost for each policy. The application program would also include transaction management software. The transaction management software would do such things as store and retrieve information from a database and pass the appropriate data back to the clients. Additionally, the application program would contain session control software. As multiple clients try to access the application, there must be control to ensure that these clients all operate properly. The session control software provides this control.

It should be noted that the user interface to the system is through the clients. A human user will input data through a client computer and call up data on the screen of the client computer. To test the application, the user interface has been the traditional point of connection to the system. Test systems that simulate the operation of the user can

input data through the user interface and determine whether the appropriate results are achieved.

Traditionally, one of a few types of tests is performed. A functional test is a test in which an input is applied and an output is observed. If the expected output is not observed, it can be inferred that the program under test is not functioning correctly. Another type of testing is called load testing. Load testing is particularly important in a client/server application where there are many clients that could be operating simultaneously. As more clients operate, the response time slows down. In a load test, the test system stimulates more client/server interactions and measures the response time. Another type of test is called a regression test. The object of a regression test is to determine whether the performance of a piece of software has changed. Such a test might be used during software development to verify that the software was not adversely impacted by a change in the program. In a regression test, a known set of inputs is applied and the results measured and compared to values that were observed at a prior time (called the baseline) when the test was run.

A more recently developed architecture for an enterprise wide application is described as an N-tier enterprise model. This model also includes clients and a server. The server does the same functions as in a traditional application, however, the specific programs providing these functions are different. One difference is that the transaction management and session are not provided by the application program. Rather, they are provided by an application server, which is more generally described as a "container." It will be appreciated that there are a finite number of functions that must be provided in order to do transaction management, such as store and retrieve data. Thus, there is no need to have application specific programs for each of these functions. The same is true for session control software. While the clients in each application might be doing different things, the need to coordinate the simultaneous use of all the clients is uniform across all applications. This function, too, is provided by the application server.

The business logic is the only function that is unique to the application. In the N-Tier enterprise model, the business logic is programmed as objects. One common programming language used for enterprise wide applications is called JAVA (a registered trademark of Sun Microsystems). While any programming language could be used, in the

description of the preferred embodiment the programming language JAVA will be referred to. Objects in JAVA that execute the business logic are often called "beans." In the description of the preferred embodiment, the objects will be referred to as JAVA beans. However, it should be appreciated that objects created in other languages (also known as software components) might be used and could be tested in the same manner as described herein. Also, it is not strictly necessary that the business logic be implemented in an object-oriented language. The term "object" will be used in conjunction with the preferred embodiment. Also, the term Enterprise Java Bean, or EJB, will be used in conjunction with the preferred embodiment. However, it should be understood that the invention is not limited to testing enterprises that are programmed with an object-oriented language.

The application server is a commercially available program. There are approximately 5 widely used and commercially available application servers. Having the application server commercially available reduces the time it takes a programmer to develop an application. The application programmer can focus only on the objects needed to program the business logic. Testing should also focus on the objects that contain the business logic.

A further benefit that has resulted from the use of application servers is that the EJBs can be re-used in different applications, which will also run on the application servers. While not all applications will use any given EJB, it is possible that an EJB will perform a function used by similar applications. For example, many applications set up to take orders from consumers will have an EJB that performs the function of receiving and validating a credit card number. Thus, an application developer might write an EJB for this function and use that EJB for every order entry application developed.

We have identified several test challenges that result from the new architecture. First, the EJB has no user interface. As a result, the standard methods of testing through the user interface are not well suited to this type of testing because the user interface would have to be simulated before testing. Secondly, the EJB is often developed with the expectation that it will operate with any application server that might be available. In theory, each application server should respond to the software that is applied to it in the same way because each application server should meet a common specification. In

practice, though, an EJB might perform differently on different application servers and might not work at all on some. It would be desirable to provide a service which tests software objects and thereby relieves the burden on a program developer to provide and support hardware and software for testing the software component or application.

SUMMARY OF THE INVENTION

With the foregoing background in mind, it is an object of the invention to provide a system that provides easy testing of software objects and to reduce the burden on a program developer for maintaining a test system. The foregoing and other objects are achieved in a system that accepts as input objects and automatically creates test drivers for these objects. The test objects are provided to a test bed comprising an application server including an Operating System and system level software libraries. The test bed is where the objects are tested by application of the test drivers.

In a preferred embodiment, the test bed comprises a collection of application servers. In accordance with another feature of the preferred embodiment, a system to create the test driver and the test bed are provided by an application service provider. Access to the test system is provided by passing a representing of the object under test to the application service provided through a network interface. According to another aspect of the preferred embodiment, the application service provider provides test services on a fee for service basis.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood by reference to the following more detailed description and accompanying drawings in which:

FIG. 1 depicts a system according to a preferred embodiment of the invention.

DETAILED DESCRIPTION

FIG. 1 illustrates that program objects, such as EJBs 300a...300c are created by an application developer 310. While the invention is described in relation to testing EJBs, it should be appreciated that other software components and applications could also

be tested in a similar manner. It is intended that these beans will ultimately be assembled into an application to be used by ultimate customer 312. Customer 312 operates a client server application that includes clients 314a...314c and a server 318. The clients are connected to the server through some form of network 316. Often the network is a TCP/IP network or some other form of Intranet. The specifics of the network are not important to the invention.

Server 318 is equipped with an application server, which is commercially available software. The developer might or might not be aware of the specific application server that will be used. However, the developer would like to be able to test the EJBs 300A...300C without limiting the testing to a specific application server and without having to create the environment that will be used by the ultimate customer 312.

Application service provider (ASP) 320 provides a service to allow testing of the EJBs 300A...300C. The EJBs are provided to the ASP 320 over network 322 or by other transport means. Network 322 is in a preferred embodiment the Internet. If the Internet is used for the connection, all of the items depicted as part of the ASP 320 can be remote from either the developer's site or the site of the ultimate customer 312. It should be appreciated that if the ASP 320 owns the software and hardware for testing, then the developer does not have to spend the money to buy and maintain this equipment and software.

Interaction between the developer and the application service provider is provided by a business interface 324. The business interface 324 ensures that the developer 310 provides the appropriate data. The business interface 324, as is known for many web based applications that charge for goods or services, also obtains credit information so that a fee can be assessed for the testing services that can be obtained from ASP 320. Business interface 324 can then do such business functions as collect a fee for the testing service provided. Business interface 324 might also provide service such as scheduling test times and otherwise managing the way developers access ASP 320.

Some of the data that must be provided to ASP 320 is the EJBs that are to be tested, the kinds of test that are to be run and instructions on how to process the results. It is not necessary that this information be provided all at once. It is likely that the exchange of information will be interactive. For example, in the initial interaction, the

EJBs to be tested might be provided. Then the tests to be run could be selected and a source of data for those tests might be indicated. Once the tests are run, the format of the output might be specified. Interactions between the developer and the ASP are described in more detail below.

5   Once the information is available to the ASP 320, test engine 326 is provided with the information. Test engine 326 creates a set of test drivers 328. There is a test driver for each test and each data set to be used in the test. The number and kinds of test drivers that are to be created are determined from the information provided by the developer.

The EJB to be tested is then loaded into a test bed 330A...330C. The system

10   provides multiple test beds, each representing one of the containers (or application servers) which with the EJB is to be used. Depending on the inputs provided by the developer, the EJB might be tested within one or more of the test beds. For purposes of explanation, testing within one test bed will be described. However, it will be appreciated that an important ability of the invention is to allow testing of a EJB in

15   multiple containers for such things as determining differences in performance in different containers or containers in which the EJB will not work.

Once the EJB under test is loaded into the test bed, the test drivers are run. The test drivers simulate the operation of clients 314A...314C. The connection to the test bed is made over a network 329. As with network 316, this could be a TCP/IP network or

20   some other type of network. It could also be the Internet. It could be that the test bed was physically on the same computer hardware as the test drivers, which would avoid the need for any physical network connection. However, because the ASP 320 could be providing test services to many developers simultaneously, it is expected that many separate computers will be used to implement the test beds 330A...330C.

25   A further service that ASP 320 might provide is a test management service. Significant effort must be expended in a development organization to track the testing that has been done. For example, for a regression test, it is necessary that a baseline is stored and the same test data used to create the baseline must be used for future regression testing. For functional testing, the data sets with the inputs and expected

30   outputs must be available. ASP 320 includes a test warehouse 334. As tests are run, information about the tests and also the data used can be stored in the test warehouse 334.

As with testing services, the test warehousing services are in a preferred embodiment provided for a fee, which business interface 324 tracks and assesses.

It was described above that tests were performed on a single EJB. It is possible that multiple EJBs are to be tested simultaneously and the developer will specify that a test will include multiple EJBs that would have to be provided to ASP 320.

However, even when an application developer thinks that a single EJB or a few EJBs are to be tested, it is possible that simply sending those EJBs will not be sufficient to allow ASP 320 to run the required tests. Object oriented programming languages often have a hierarchy. Objects are included within other objects at a higher level in the hierarchy. Objects at the lower level of the hierarchy can "inherit" attributes from objects that are at higher levels in the hierarchy. For example, an EJB might use methods (which are executable code) or properties (which are like variables) that have been inherited from an object at a higher level of the hierarchy. Thus, if only the lower level EJB were passed to ASP 320, it would not have sufficient information to conduct the test on that EJB. Specifically, it would lack the methods and properties defined at the higher levels of the hierarchy.

To enable ASP to conduct testing on an EJB, it must have access to the higher level EJBs from which that EJB has inherited attributes. To provide this access, the business interface 324 downloads to the developer a packer 332. Packer 332 is a program resident on the developer's computer that can find on the computer memory certain EJBs and send them to ASP 320. Packer 332 knows which EJBs to obtain because each EJB includes a portion that describes the dependencies of that EJB. In JAVA, an EJB is represented as a .jar (Java Archive) file. The .jar file contains the executable code of the EJB in addition to a description of the dependencies of the file. In this way, business interface 324 can figure out all the EJBs that are needed to test a specific EJB. These EJBs could be uploaded to ASP 320 for the test. While a .jar file is described in relation to EJBs, it should be understood that any archive relating to the software component being employed could also be used.

For functional testing of the EJBs, the data for the tests can come from one of several different sources. One possibility is that the user can provide the data sets for the tests. Each data set would specify inputs and expected outputs for a particular method. A

test driver would be constructed that would invoke that method with the specified input data. When the test engine ran that driver, it could compare whether the expected data were observed.

It is also possible that the test data could be obtained in an automated or semiautomatic fashion. Each Java bean contains a public interface that specifies the methods and properties of that bean. From the interface, it is possible to know which properties must be provided and measured to test a particular method. Thus, a table could be constructed for each method, listing the inputs and the outputs encountered in executing that particular method. A data set for testing the method would have to have data values for each entry in that table.

Once the data table is generated, the inputs could be filled in automatically in a random fashion. Test generation software, such as TestMaster sold by Teradyne Software and System Test of Nashua, New Hampshire could be used to create the expected outputs. TestMaster is a program that receives a description of a system (including a software object) under test. It can generate a set of inputs that should test the system and, based on the description of the system that has been input, can also determine an expected output.

However, there are many types of tests that can be done without knowing the expected output value. For example, a regression or load test does not need to know the "right" output value. For regression testing, it only matters that the value is same or different than the baseline. For load testing, it matters how long it took the output to be provided. Another test than can be done without knowing the expected outputs is a check to see that the appropriate input was properly recorded by the EJB. As a further example, some tests are simply intended to track the execution of the EJBs. Test engine 326 can access test beds 330A...330C to determine which methods were accessed during a test and can also determine whether a particular test driver invoked a bean that executed to normal completion. For example, test engine 326 would know whether a particular method terminated because of an error or exception condition – which might signify a problem with the EJB. To run this type of test, the expected values of the outputs are also not required to be known. Rather, occurrences during the invocation of the bean that are

the useful information desired by the developer. Thus, it might not be necessary to generate expected outputs to perform the required tests.

However, where it is necessary to perform a functional test without an automated test generation tool being available, it is possible that the developer will provide the expected output values. ASP 320 could provide a template data table for each method of the EJB under test. This would tell the data values that the developer would need to provide to test the EJB. By automatically generating the data tables from the information in the bean's .jar file, the developer knows the minimum set of data values that need to be provided, thereby reducing the test creation time. The ASP 320 could generate the inputs and have the developer specify the appropriate outputs or the developer could specify both the inputs and the outputs to the test.

An example of a test session for a particular embodiment will be described. A typical test session would be initiated by a developer "logging on" to the ASP web site. Each developer would have an account with the ASP, so that space on the ASP computers could be allocated to the storage of their test information and to enable appropriate billing. During the registration process, the ASP would seek permission to download the packer 332 to the developer's computer and install that packer.

The developer would transfer the bean or beans to be tested to the ASP by uploading the .jar files for these EJBs. A user interface from ASP 320 would guide the developer through the provision of the data needed to specify the tests to be performed. The user interface would also guide the developer through identifying which data to use to develop test drivers. The developer might specify a data set from the test archive or might provide the data set or might specify that the ASP should generate the data automatically. A developer might also specify the containers (or application servers) with which to test the EJBs. Data tables could have data values, references to other data tables, and/or program code fragments.

The ASP 320 would create the test drivers and prepare them in a format for a human to read them. The test drivers would then be provided to the developer. The developer might then edit the test drivers. The developer might put what is the equivalent of "break points" in the test drivers. When a break point is encountered, the test driver would either stop execution or would record values of variables or execution state. The

developer might specify exception or error handling conditions. The developer would choose the edits to make based on knowledge to the EJB under test and the information that is expected to be generated by the testing. The modified drivers would be uploaded to ASP 320.

5   When the developer had specified the required information, the developer would initiate the test. During execution, the ASP would perform the required steps automatically. It would deploy the EJBs (which means that they would be loaded into the application servers in the ASP). The ASP would also invoke the test drivers. If necessary, the ASP would schedule the tests to be run. As with other web sites shared by

10   multiple users, it would be necessary to ensure that each test were run at a time when hardware resources were available and so, depending on the load, some scheduling might be required.

Once the test drivers had been executed, the results of the test would be available. The developer would specify which results would be displayed and their formats. For

15   example, for functional testing, the developer would likely view results in a pass/fail format. For load testing, the developer would probably specify a graphical output, showing response time as a function of the number of EJBs simultaneously executing. Alternately, results for load testing might be viewed in a pass/fail format as measured against specification limits.

20   Once the sessions were completed, ASP 320 would tally the operations performed and compute and send a bill for the services actually used.

It will be appreciated that the ASP 320 is very similar to many other web based applications. It has a user interface that uploads and downloads code from a user. The specific application on ASP 320 is a test application. While many of the specific test

25   capabilities described above have never heretofore been done, having read the foregoing description and attached materials, one of skill in the art will understand how to implement ASP 320.

Having described one embodiment, numerous alternative embodiments or variations might be made. For example, various servers are described. The servers might

30   be located on different pieces of computer hardware or on the same pieces of computer hardware.

Also, a service for application developers was described. The service might be used by application integrators or even the ultimate customer for software maintenance, life cycle testing, or as part of an acceptance or qualification procedure.

As another example, it was described that certain functions were performed by the business interface. The functions might be performed by different pieces of code or code resident on different computers. For example, it is not necessary that business interface 324 determine which EJBs must be provided along with the EJB under test. This function could be performed by test engine 326 or by packer 332. The overall functioning of the system would be the same.

Particularly, it was described that ASP 320 is located remotely from the site of the ultimate customer and the developer. In a preferred embodiment ASP 320 is accessed over the Internet. However, some developers might prefer not to transmit their code under development over the Internet. In that case, the functions described as part of ASP 320 would be installed on a server at the developer's site. In that case, net 322 would be the developer's internal network rather than the Internet.

Also, it was described that a data table was constructed to describe the parameters in a data set. It will be appreciated that the table is conceptual. The data might be physically stored in any kind of data structure and might be stored in some other way – such as in a text file instead of a data structure. Object recording is a method of saving and replaying user defined data structures.

Further, certain types of tests were described above. The list is intended to be illustrative rather than limiting. ASP 320 might provide other types of tests. It might provide a compliance test – for example to verify that each EJB is in compliance with the specifications for objects written in JAVA or appropriate language. Alternatively, ASP 320 might test to determine whether a particular EJB uses any features that are not supported by a particular application server.

Having described preferred embodiments of the invention it will now become apparent to those of ordinary skill in the art that other embodiments incorporating these concepts may be used. Additionally, the software included as part of the invention may be embodied in a computer program product that includes a computer useable medium. For example, such a computer usable medium can include a readable memory device,

such as a hard drive device, a CD-ROM, a DVD-ROM, or a computer diskette, having computer readable program code segments stored thereon. The computer readable medium can also include a communications link, either optical, wired, or wireless, having program code segments carried thereon as digital or analog signals. Accordingly, it is 5 submitted that that the invention should not be limited to the described embodiments but rather should be limited only by the spirit and scope of the appended claims.

CLAIMS

What is claimed is:

1   1.   A method of providing a software testing service comprising the steps of:

2   receiving, by a service provider, at least one software component from a customer;

3   providing, by said service provider, test code for testing said at least one software

4   component;

5   testing with said test code said at least one software component;

6   monitoring, by said service provider, results of said testing; and

7   providing, by said service provider, the results of said testing to said customer.

1   2.   The method of claim 1 further comprising the step of charging said customer for

2   providing said software component testing service.

1   3.   The method of claim 2 wherein said step of charging is performed on a per session

2   basis.

1   4.   The method of claim 2 wherein said step of charging is performed on a periodic

2   basis.

1   5.   The method of claim 1 wherein said step of receiving is done over a network.

1   6.   The method of claim 5 wherein said step of receiving is done over a TCP/IP

2   network.

1   7.   The method of claim 5 wherein said step of receiving is done over the Internet.

1   8.   The method of claim 1 wherein said step of receiving further comprises the step of

2   receiving directions regarding the type of testing to be done on said software component.

1   9.   The method of claim 8 wherein said step of receiving directions regarding the type

2   of testing to be done is selected from the group consisting of functional testing, regression

3   testing, load testing and compliance testing.

1   10.   The method of claim 1 wherein said step of receiving further comprises the step of

2   receiving directions regarding how to process results from said testing.

1   11.   The method of claim 1 wherein at least a portion of the results from said testing is

2   provided in a pass/fail format.

1   12.   The method of claim 1 wherein at least a portion of the results from said load

2   testing are provided in a graphical format.

1   13.   The method of claim 1 wherein said software component comprises an EJB.

1   14.   The method of claim 1 wherein said software component comprises an

2   application.

1   15.   The method of claim 1 wherein said software testing service is provided as a web

2   site.

1   16.   The method of claim 1 wherein said step of providing further comprises the step

2   of providing access to multiple containers for testing the performance of said software

3   component in different ones of said multiple containers.

1   17.   The method of claim 1 wherein said step of providing further comprises the step

2   of determining the parameters of the data sets to test a software component by examining

3   the interface of each method in the software component and determining what inputs and

4   outputs must be specified for that software component.

1   18.   The method of claim 1 wherein said step of testing further comprises the step of

2   testing program components typically used through a graphical user interface by building

3   test drivers for the components and testing without the graphical user interface.

1   19.   The method of claim 1further comprising the step of using a test warehouse to

2   store test data for customers for later use by the customers.

ABSTRACT OF THE DISCLOSURE

A system that provides easy testing of software objects and reduces the burden on a program developer for maintaining a test system is presented. The system accepts as an input objects and automatically creates test drivers for these objects. The test objects are

5   provided to a test bed comprising an application server where the objects are tested by application of the test drivers.

In a preferred embodiment, the test bed comprises a collection of application servers. An application service provider provides the system test driver and the test bed. Access to the test system is provided by passing a representing of the object under test to

10   the application service provided through a network interface. The application service provider provides test services on a fee for service basis.
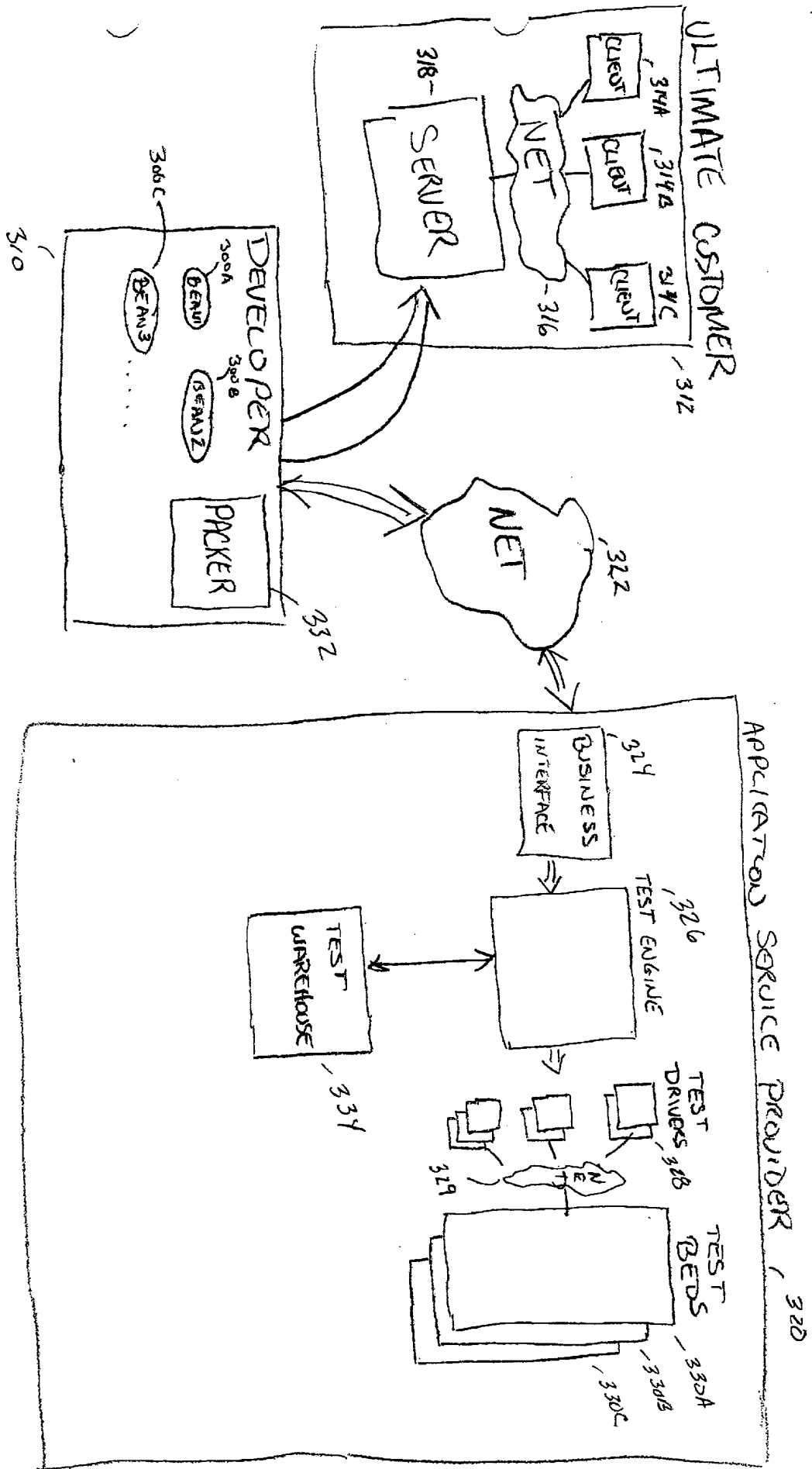
FIG. 1